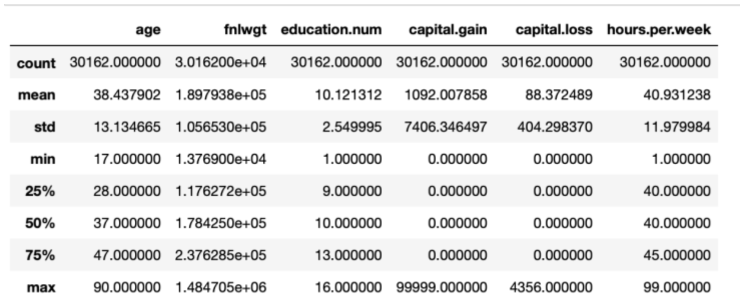
Part 1 – Building a Basic NN

The income dataset is used to predict whether the income of an individual is above or below 50K based on certain variables. It contains 32,561 rows and 15 columns, with rows representing the number of entries and the columns representing variables. The last column is the income variable which shows if the individual makes above or below 50k. Some of the variables used to determine the income are age, work class, education, occupation, marital status, race, sex, capital gain, capital loss, native country.

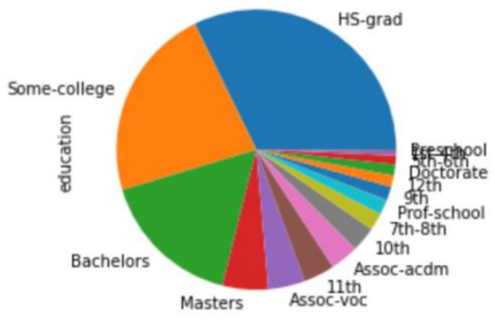
The dataset has both categorical and numerical variables. Some of the rows had unwanted characters like "?" which were removed. Some statistics for the numerical variables are shown in the table below.



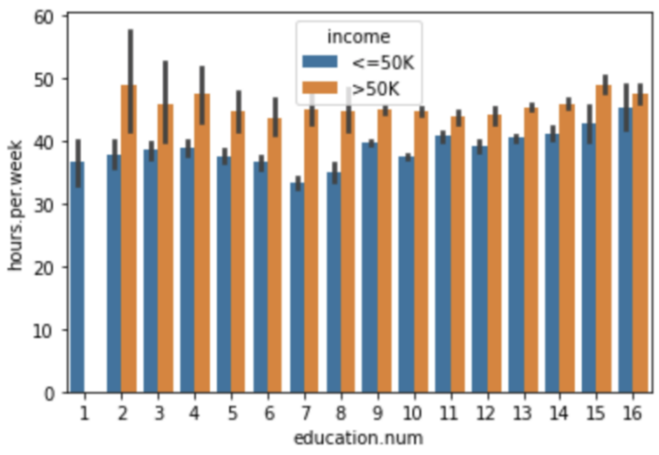
	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	30162.000000	3.016200e+04	30162.000000	30162.000000	30162.000000	30162.000000
mean	38.437902	1.897938e+05	10.121312	1092.007858	88.372489	40.931238
std	13.134665	1.056530e+05	2.549995	7406.346497	404.298370	11.979984
min	17.000000	1.376900e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.176272e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.784250e+05	10.000000	0.000000	0.000000	40.000000
75%	47.000000	2.376285e+05	13.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

The table shows that the minimum age of individuals in the dataset is 17 and the maximum age is 90. "fnlwgt" stands for final weight which represents an estimate of individuals who belong in the same set of combination in each row. Education number stands for the years of education for each individual with minimum being 1 year and maximum 16 years. The minimum for both capital gain and loss is 0 and maximum value is $7,407 and $404. Hour per week is the number of hours an individual works per week and the minimum is 1 hour and maximum is 99 hours.

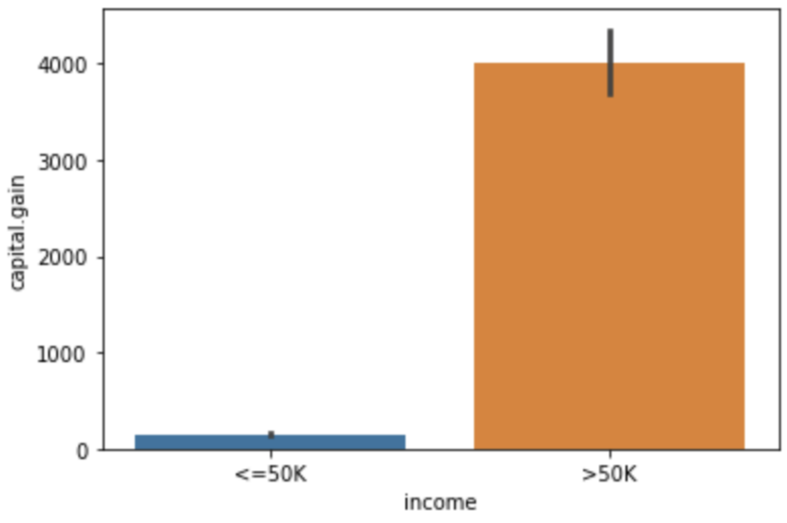
Data Visualization



The plot shows the educational level of the individuals in the dataset. Majority of the individuals are HS graduates and the second and third highest educational level are "some college" and "Bachelors".

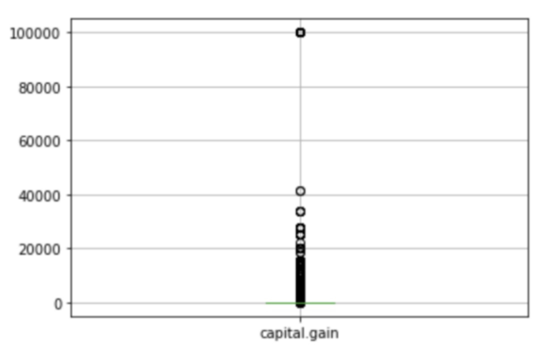


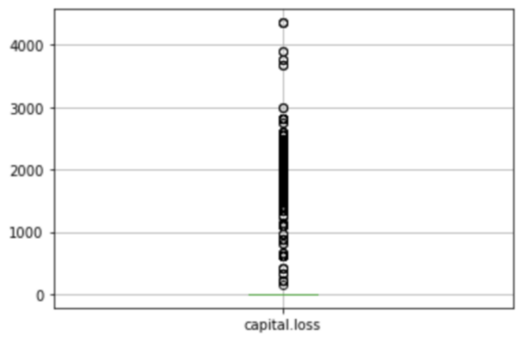
From this plot, we can infer that those individuals who worked more hours a week made more than 50K and individuals who worked less generally made less or equal to 50k. There's no correlation between education number and income.



The plot shows that most individuals with a higher capital gain made more than 50k.

Data Preprocessing





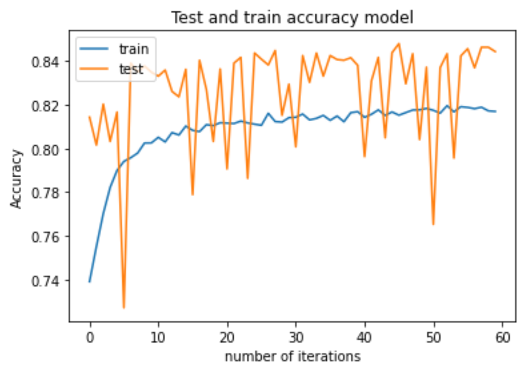
To obtain a higher accuracy for our model, we removed the columns capital gain and loss variable because of it’s extreme amount of outliers as shown in the box plot above. We normalized the final weight variable and one hot encoded the categorical variables. One hot encoding creates a new columns for each unique category in a variable. This resulted in an increase in the number of columns in our dataset which also helped in increasing the accuracy of model.

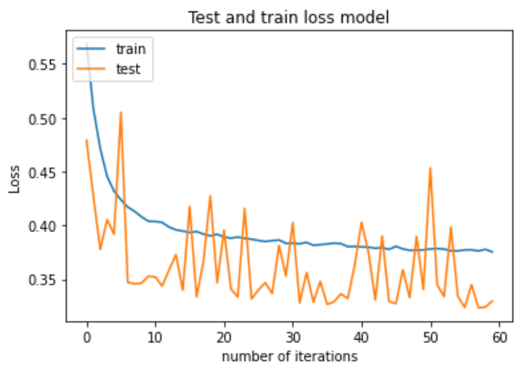
Structure of Neural Network

To create our neural network, we first divided our preprocessed data into test and training data. 80% of the data was used to train the model and the leftover 20% was used to test the model. Our target for the model is to predict the income (above or below 50K) of the individuals.

Our model consists of three hidden layers and an output layer. There are 102 input variable in the model. The first hidden layer has 64 nodes and uses relu activation function. The second hidden layer has 32 nodes and uses relu activation and the third layer has 16 nodes and uses relu acitivation function. The output layer has 1 node and uses sigmoid activation function.

We achieved a test accuracy of **84.43%** and a train accuracy of **81.87%.** The graphs for the accuracy and loss model can be seen below





<u>**Part 2 – Optimizing NN**</u>

Modifying Hyperparameters

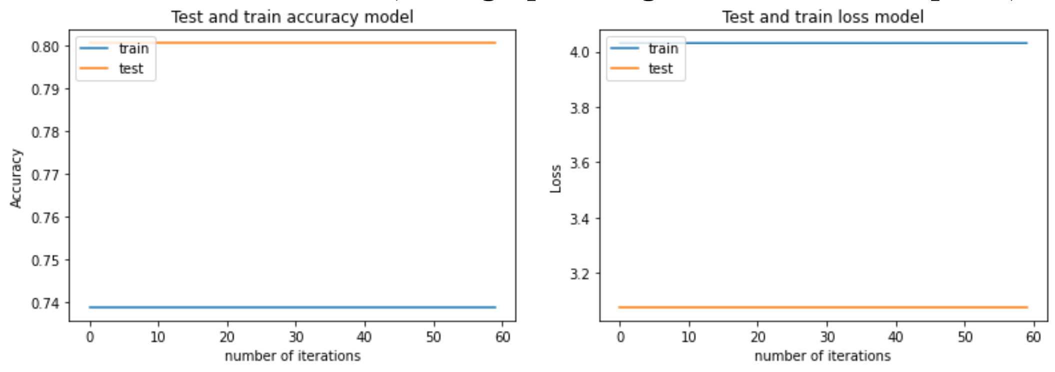
The next step was to optimize the model by tuning the hyper parameters. The next following tables and graphs are the parameter details for every hyper parameter that's being tuned and the corresponding accuracy and loss graphs for the test and train data.

Step 1

Activation Function Tuning

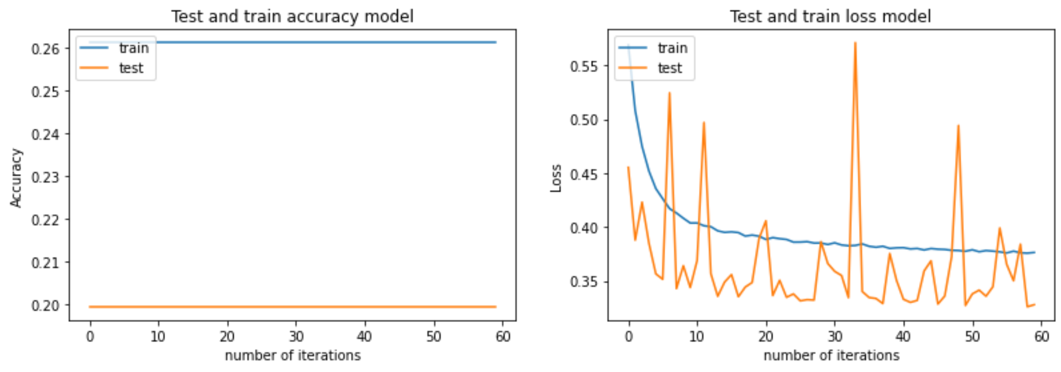
Activation Function	Optimizer	Dropout	Test Accuracy
sigmoid	SDG	NA	84.43
Tanh	SDG	NA	80.05
Softmax	SDG	NA	19.94

Activation function "Tanh" (Note: graph for sigmoid can be seen in part I)



We can see that by using the Tanh activation function for the output layer, we get a constant accuracy for all the iterations performed.

Activation function "softmax"



Using SoftMax as the activation function gave us really poor results. After comparing the accuracy for all three activation functions, we decided to go with SGD as it performs the best amongst all three with an accuracy of **84.43%**

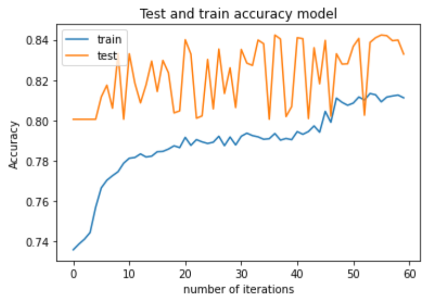
Step 2

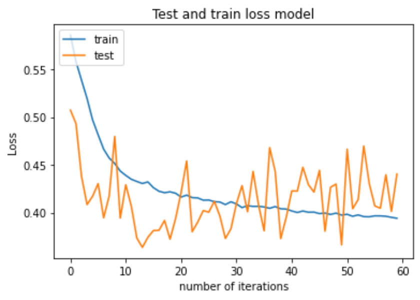
Dropout Tuning

Activation Function	Optimizer	Dropout	Test Accuracy
Sigmoid	SGD	0.5	83.29
Sigmoid	SGD	0.2	84.30
Sigmoid	SGD	0.02	84.48

A drop out layer was added after the second hidden layer with a drop out value of 0.5

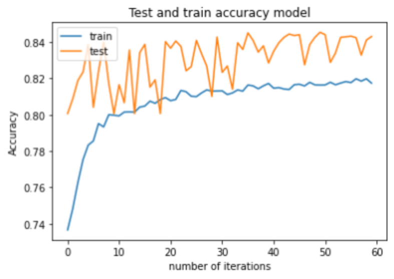
Drop out value 0.5

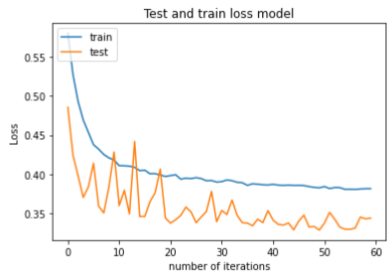




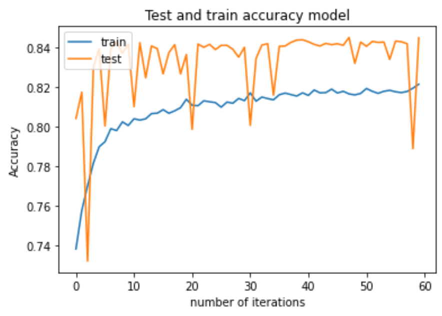
We can see that adding a drop out value reduced the accuracy, so we decided to reduce the drop out value in the next iterations.

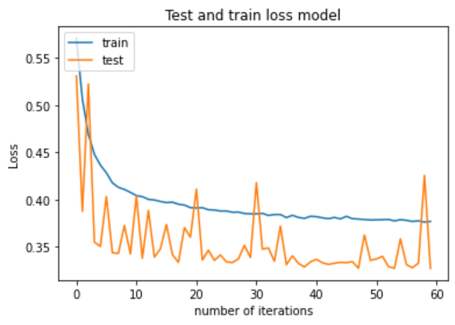
Drop out value 0.2





Drop out value 0.02





After reducing the drop out value to 0.02, we achieved an accuracy of **84.48%** which was the best amogst all the other drop out values. So we fixed this as our final drop out value.

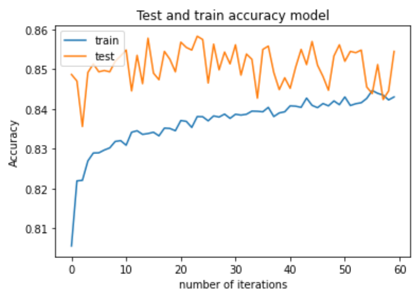
Step 3

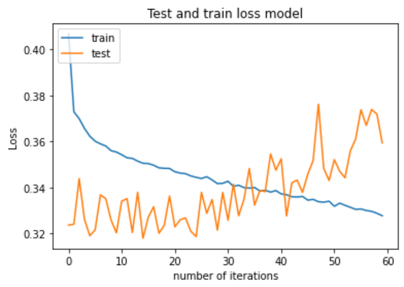
Optimizer Tuning

Activation Function	Optimizer	Dropout	Test Accuracy
Sigmoid	Adam	0.02	85.44
Sigmoid	RMSProp	0.02	85.44
Sigmoid	Nadam	0.02	85.28

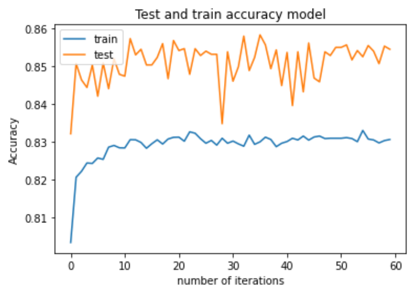
The next step was to tune the optimizer parameter.

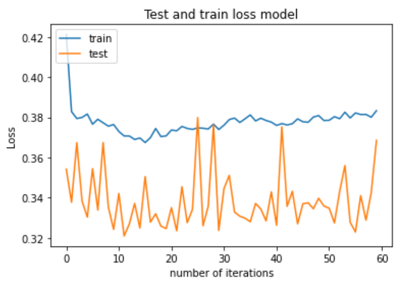
Optimizer "Adam"



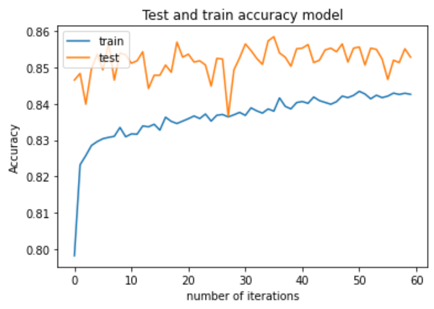


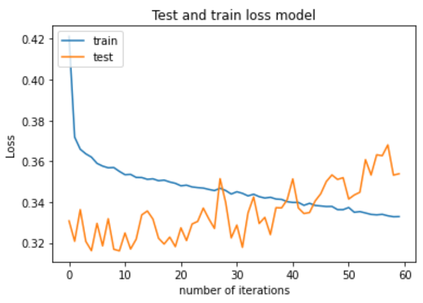
Optimizer "RMSprop"





Optimizer "Nadam"



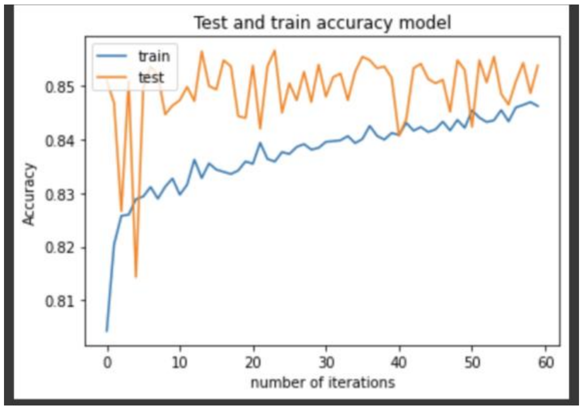


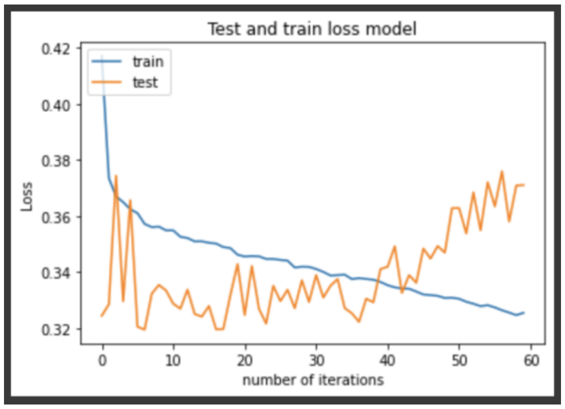
From the above hyper parameter tunings, the best accuracy is achieved with our optimizer being "Adam". The test accuracy is **85.44%** and the train accuracy is **84.54%**

Base Model

After modifying the hyperparameters, we modified our original NN model to create our base NN model using the set up that returned the best accuracy. Our input layer, hidden layers and output layer from our original NN model were not changed. Sigmoid gave the best accuracy which we left as the activation function for the output layer. We added a dropout of 0.02 after the second hidden layer and changed the optimizer to “Adam” because they both returned the best accuracy.

All together, our base model consists of an input layer which expects 102 input variables. The first hidden layer has 64 nodes and uses relu activation function. The second hidden layer has 32 nodes and uses relu activation. A dropout layer of 0.02 is added after the second hidden layer and the third layer has 16 nodes and uses relu acitivation function. The output layer has 1 node and uses sigmoid activation function. The model uses “Adam” optimizer.





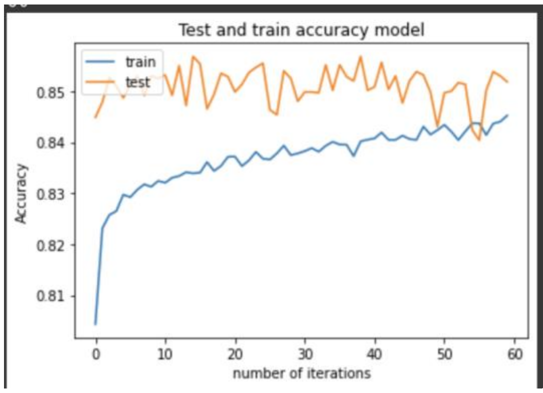
Our base model gemerated a train accuracy of **85.00%** and test accuracy of **85.38%** which is higher than our original model. The original model had a train accuracy of 81.87% and test accuracy of 84.43%.

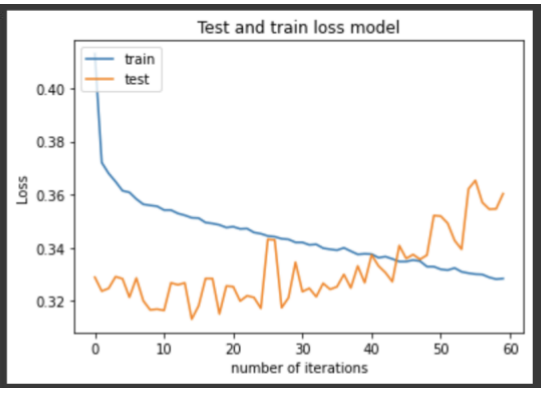
Methods to Improve Model

After creating the base model, we tried different deep learning methods that can increase the training speed and improve the accuracy of our base model. Some of the methods we tried were early stopping, regularization methods, K fold and batch normalization.

Early stopping

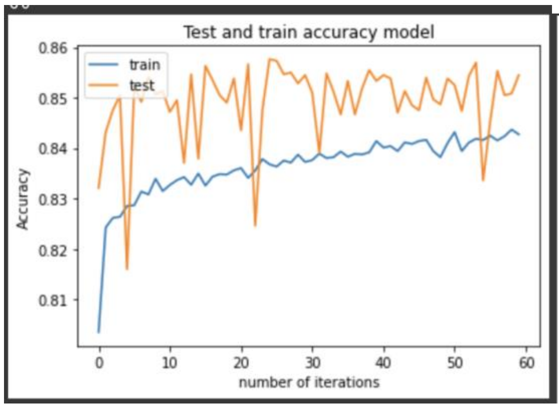
Early stopping is used reduce overfiitting in the model without affecting the accuracy. It stops training the model once the model performace stops improving on the test dataset. We added two arguments to the early stopping call back, “monitor” and “patience”. Monitor allows us to specify which quantity to measure in order to trigger early stopping. In our case we chose to monitor our loss. “Patience” is used to specify the number of epoch with no improvement after which the training will be stopped.
Adding early stopping callback to our model generated a training accuracy of **84.24%** and test accuarcy of **85.18%.**

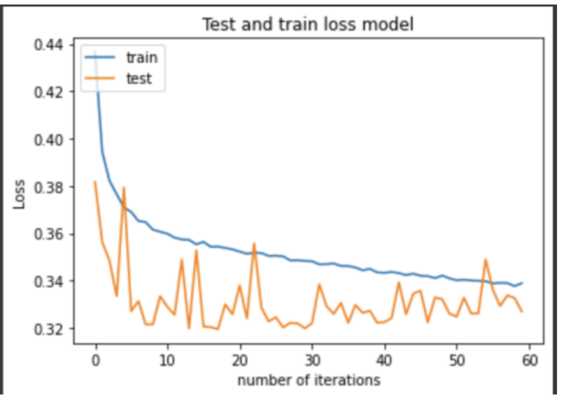




Adding regularization

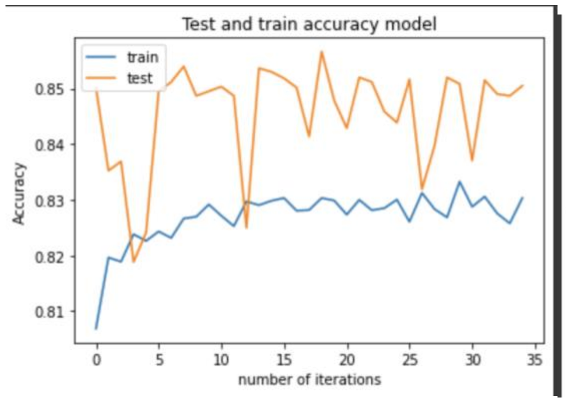
Regularization also helps to reduce overfitting of the model in the training data by adding a penalty on the weight size of the layers. We added the L2 regularizer of 0.001 to the 2nd and 3rd dense layer of our neural networks. L2 is also known as ridge regression, this adds the sum of the squared weights as the penalty term. After adding the L2 regularizer to our model, it increased the train and test accuracy to **84.50%** and **85.45%** respectively.

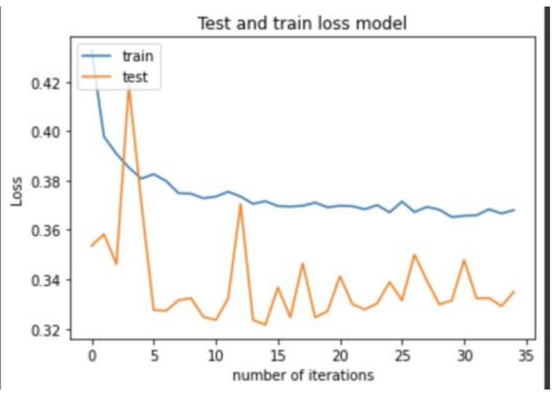




Batch Normalization

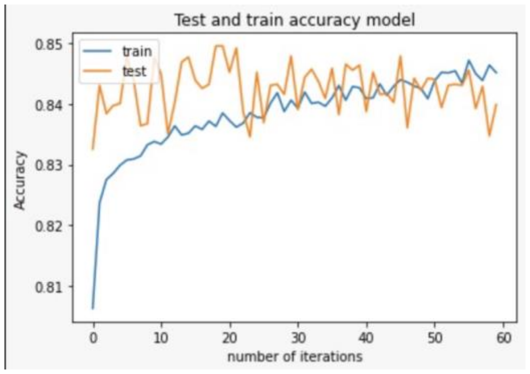
Batch normalization makes our model faster and more stable by accelarate the training process of through normalization. It fixes the mean and varince of each layer's input for each mini batch by rescaling and re-centering the inputs as they enter each layer. Batch normalization also gives room for higher learning rate and it also has a regularizing effect. Adding the batch normalization layer generated an accuracy of **83.75%** for the train and **85.05%** for the test, which is a bit lower than our previous accuracy with the L2 regularizer.

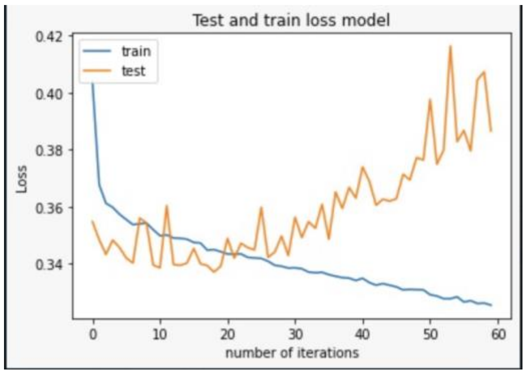




K-Fold cross validation

K fold cross validation helps with the training procedure. The k parameter species the number of folds the dataset will de divided into and each fold appears in the training set. This enables the model to learn the data distribution better. This generated a test accuracy of **83.99%** and train accuracu of **84.86%**


Test and train accuracy model
train
test
Accuracy
number of iterations


Test and train loss model
train
test
Loss
number of iterations